Q1. In Python 3.X, what are the names and functions of string object types?

In Python 3.x, there are two types of string objects: str and bytes.

The str type represents a string of Unicode characters and is used for working with text. It supports all the usual string operations like concatenation, slicing, and indexing.

The bytes type, on the other hand, represents a sequence of bytes and is used for working with binary data, such as images or audio files. It supports similar operations to str, but the results are in terms of bytes rather than characters.

Q2. How do the string forms in Python 3.X vary in terms of operations?

* **<stirng>.isdecimal() ->** Returns True if all characters in a string are decimal.
* **<string>.isalnum() ->** Returns True if all characters in the string are AlphaNumeric.
* **<string>.istitle() ->** Returns True if first character in a string is in Uppercase.
* **<string>.partition(<sub\_string>) ->** Splits string at first occurance of sub string and returns a tuple of 3 elements.
* **<string>.rpartition(<sub\_string>) ->** Splits string at last occurance of sub string and returns a tuple of 3 elements.
* **<string>.isidentifier() ->** Returns True if give string is a valid identifier name.
* **len(<string>) ->** Returns the length of the given string.
* **<string>.index(<sub\_string>) ->** Returns the lowest index of substring if substring is found in the string.
* **<string>.rindex(<sub\_string>) ->** Returns the highest index of substring if substring is found in the string.
* **max(<string>) ->** Returns the highest Alphabetical Character in the string as per ASCII.
* **min(<string>) ->** Returns the lowest Alphabetical Character in the string as per ASCII.
* **<string>.splitlines() ->** Returns a list of lines in the string.
* **<string>.capitalize() ->** Returns the string with first character capitalized.
* **<string>.upper() ->** Returns the string with all characters in uppercase.
* **<string>.lower() ->** Returns the string with all characters in lowercase
* **<string>.casefold() ->** Returns the string in lowercase which can be used for caseless comparisions.
* **<string>.expandtabs(no\_of\_spaces) ->** Replaces tabs in a string with specified no of spaces default is 8
* **<string>.find(<sub\_string>) ->** Returns lowest index of substring if substring is found in the string else returns -1.
* **<string>.rfind(<sub\_string>) ->** Returns highest index of substring if substring is found in the string else returns -1.
* **<string>.count(<char>) ->** Returns the no of occurances of the char in the given string.
* **<string>.split(<sep>) ->** Returns list of words seperated by given sep else seperated by whitespace.
* **<string>.rsplit(<sep>) ->** Returns list of words seperated by given sep else seperated by whitespace scanning from end.
* **<string>.lstrip() ->** Returns a copy of where leading whitespaces are removed.
* **<string>.rstrip() ->** Returns a copy of where trailed whitespaces are removed.
* **<string>.strip() ->** Returns a copy of where both leading and trailing whitespaces are removed.
* **<string>.swapcase() ->** Swaps lowercase characters with uppercase and vice versa.
* **<sep>.join(<list>) ->** Concatenates a list or tuple of words with intervening occuernces of sep.
* **<string>.translate(<mapping\_table>) ->** translates the characters using table.
* **<string>.maketrans(<dict>) ->** Creating a mapping translation tbale usable for **<string>.translate(<mapping\_table>)**
* **<string>.replace(<char\_1>,<char\_2>) ->** Replace all occurances of char\_1 with char\_2 in string.
* **<string>.encode() ->** Encodes string into any encoding supported by python. Default encoding is **UTF-8**.
* **<string>.ljust(<no\_of\_spaces>) ->** Left-justify in a field of given width.
* **<string>.rjust(<no\_of\_spaces>) ->** Right-justify in a field of given width.
* **<string>.center(<no\_of\_spaces>) ->** Center-justify in a field of given width.
* **<stirng>.zfill(<length>) ->** Zfill adds zeros to the begining of string until the specified length is reached.

Q3. In 3.X, how do you put non-ASCII Unicode characters in a string?

The **unidecode** library contains a function that takes a string object, possibly containing non-ASCII characters, and returns a string that can be safely encoded to ASCII:

from unidecode import unidecode

>>> unidecode('kožušček')

'kozuscek'

Q4. In Python 3.X, what are the key differences between text-mode and binary-mode files?

|  |  |
| --- | --- |
| **Text-mode** | **Binary-mode** |
| Text-mode files handle data as strings of Unicode characters | Binary-mode files handle data as a sequence of bytes |
| Text-mode files use the platform-specific newline convention when writing, and translate it to the universal newline convention ("\n") when reading | Binary-mode files do not perform any newline translation |
| Text-mode files automatically encode Unicode strings to bytes when writing, and decode bytes to Unicode strings when reading, using the specified encoding | Binary-mode files do not perform any encoding or decoding |
| Text-mode files provide additional high-level I/O methods for working with strings, such as readline() and writelines() | Binary-mode files provide lower-level I/O methods that work with bytes, such as read() and write() |
| Text-mode files may raise exceptions when encountering encoding or decoding errors | Binary-mode files may raise exceptions when encountering I/O errors or reaching the end of file. |

.

Q5. How can you interpret a Unicode text file containing text encoded in a different encoding than your platform's default?

Using the encoding parameter of the open() function to specify the encoding.

Q6. What is the best way to make a Unicode text file in a particular encoding format?

To make a Unicode text file in a particular encoding format in Python, you can use the open() function with the appropriate encoding argument.

Q7. What qualifies ASCII text as a form of Unicode text?

Unicode is a superset of ASCII, which means that all ASCII characters are included in Unicode. Therefore, any text that contains only ASCII characters is also a valid Unicode text.

Q8. How much of an effect does the change in string types in Python 3.X have on your code?

The change in string types in Python 3.X can have a significant effect on your code, depending on how your code interacts with strings. One of the main differences in Python 3.X is that strings are Unicode by default, whereas in Python 2.X, strings were typically encoded as ASCII or another byte-based encoding. This means that in Python 3.X, your code may need to use different string methods and encoding/decoding functions to work with strings in a Unicode context.